Tactical Security
Improving AppSec Coverage with Fewer Resources

Rich Newman, CISSP, Sales Engineer, Synopsys Software Integrity Group
May 2023
Rich Newman

13 years in software development

8 years field engineering,
Wind River Operating Systems and Tools

6 years field engineering,
Wind River Test Management

11 years (minus 12 days) field engineering,
Coverity and Synopsys

rnewman@synopsys.com
949.466.5283
Today’s Agenda

1. Software Development Process
2. SAST
3. SCA
4. DAST (IAST/Fuzz)
5. ASOC
6. Help!

- SAST - Static Application Security Testing
- SCA - Software Composition Analysis
- DAST - Dynamic Application Security Testing
- IAST - Interactive Application Security Testing
- Fuzz - Malformed protocol testing
- ASOC - Application Security Orchestration and Correlation
Software Development Process

Design
- Product Requirements
- Planning
- Cybersecurity Requirements
- Architectural Design
- Implementation

Planning
- Design Phase Verification
- SW Testing / Test Phase Verification

Architecture
- DAST / IAST
- SCA
- SAST

Automation
- System/Feature Testing
- Test Phase Verification

Education
- ASOC
- Fuzz

Validate
- ASOC
Today’s Agenda

1. Software Development Process
2. SAST
3. SCA
4. DAST (IAST/Fuzz)
5. ASOC
6. Help!
Simple Web Application

Dashboard

© 2015, Kaleidosoft Labs
What are the Components?

<table>
<thead>
<tr>
<th>Component</th>
<th>Code Lines (LOC)</th>
<th>Outstanding</th>
<th>Dismissed</th>
<th>Fixed</th>
<th>Issue Density</th>
</tr>
</thead>
<tbody>
<tr>
<td>webapp-resources</td>
<td>907</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>unusedspringsecurity</td>
<td>397</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>10.08</td>
</tr>
<tr>
<td>test</td>
<td>278</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>10.79</td>
</tr>
<tr>
<td>servlet</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>service</td>
<td>232</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4.31</td>
</tr>
<tr>
<td>model</td>
<td>555</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1.80</td>
</tr>
<tr>
<td>interceptor</td>
<td>42</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>142.86</td>
</tr>
<tr>
<td>framework</td>
<td>246</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>core</td>
<td>258</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>31.01</td>
</tr>
<tr>
<td>controller</td>
<td>333</td>
<td>13</td>
<td>0</td>
<td>0</td>
<td>39.04</td>
</tr>
<tr>
<td>auth</td>
<td>120</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>33.33</td>
</tr>
<tr>
<td>Other</td>
<td>93744</td>
<td>42</td>
<td>0</td>
<td>0</td>
<td>0.45</td>
</tr>
</tbody>
</table>
What are the risks?

Many issues in a small area of the application may indicate a design concern (fever).

Too many issues reveals high technical debt (general malaise).
Writing invalidated user input to log files can allow an attacker to forge log entries or inject malicious content into the logs:

- Injection of misleading events
- Injection of XSS attacks, hoping that the malicious log event is viewed in a vulnerable web application
- Injection of commands that parsers (like PHP parsers) could execute
Technical Debt

Any code that decreases agility as the project matures

Agility decreases as additional time is needed to investigate, resolve and test delayed issue resolution

Another task on the things to do list: Develop an inaccessible properties file
IaC – Infrastructure as Code

/**
 * Logs out a user by deleting the session
 *
 * @param userDTO
 * @param request
 */
@RequestMapping(value = "/logout", method = RequestMethod.DELETE)
public ResponseEntity logout(@RequestBody UserDTO userDTO) {
    return APIResponse.toOkResponse(“success”);
}

private String decryptPassword(UserDTO userDTO) throws NoSuchPaddingException, NoSuchAlgorithmException, InvalidAlgorithmParameterException, InvalidKeyException, InvalidKeySpecException
    String passPhrase = “b83w52KcmfP890S9010Kx8135”;
    String salt = userDTO.getSalt();
    String iv = userDTO.getIv();
    int iterationCount = userDTO.getIterations();
    int keySize = userDTO.getKeySize();
Static Application Security Testing

1. Analyze the entire application
2. Divide and conquer by components
3. Understand the risk of the issues identified
4. Set goals; technical debt, issue density, OWASP Top 10, etc. KPIs
Today’s Agenda

1. Software Development Process
2. SAST
3. SCA
4. DAST (IAST/Fuzz)
5. ASOC
6. Help!
Automating Risk Awareness

There are 25 components with security risk, how do we prioritize?
Jackson-databind Deserialization Remote Code Execution (RCE)

Triggered policy Exploit:
✓ Known exploit
✓ Zero-click RCE
✓ CVE >= 9
Deserialization of untrusted user data in Jackson Databind could allow an attacker to perform Remote Code Execution via specially crafted JSON input.

This issue exists because of an incomplete fix for CVE-2017-7525 which the vendor tried to address through an incomplete blacklist.

**Zero-click Remote Code Execution**

This vulnerability can result in the execution of code on the system, triggered by a remote attacker without requiring or relying on any third party action.

**How to fix it**

**Solution - Fix Available**

Fixed in 2.7.9.2, 2.8.11 and 2.9.4 by this commit.

Fixed in 2.6.7.3 by this commit.

Although this resolves the issue relating to Spring libraries present on the classpath, another attack vector exists (involving C3PO libraries) which is described in BDSA-2018-0788.
Vulnerabilities Found at CyRC

<table>
<thead>
<tr>
<th>CVE</th>
<th>BSDA</th>
<th>Product</th>
<th>Researcher</th>
<th>Tool</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2021-43175</td>
<td>BSDA-2021-3657</td>
<td>goautodial.goAPI</td>
<td>Scott Tolley</td>
<td>Seeker</td>
<td>Synopsys advisory</td>
</tr>
<tr>
<td>CVE-2021-43176</td>
<td>BSDA-2021-3656</td>
<td>goautodial.goAPI</td>
<td>Scott Tolley</td>
<td>Seeker</td>
<td>Synopsys advisory</td>
</tr>
<tr>
<td>CVE-2021-43177</td>
<td>BSDA-2021-2845</td>
<td>Nagios XI</td>
<td>Scott Tolley</td>
<td>Seeker</td>
<td>Synopsys advisory</td>
</tr>
<tr>
<td>CVE-2021-43179</td>
<td>BSDA-2021-2847</td>
<td>Nagios XI</td>
<td>Scott Tolley</td>
<td>Seeker</td>
<td>Synopsys advisory</td>
</tr>
<tr>
<td>CVE-2021-43179</td>
<td>BSDA-2021-2846</td>
<td>Nagios XI</td>
<td>Scott Tolley</td>
<td>Seeker</td>
<td>Synopsys advisory</td>
</tr>
<tr>
<td>CVE-2021-22116</td>
<td>BSDA-2021-1329</td>
<td>RabbitMQ</td>
<td>Jonathan Knudsen</td>
<td>Defensics</td>
<td>Synopsys advisory</td>
</tr>
<tr>
<td>CVE-2021-93176</td>
<td>BSDA-2021-1608</td>
<td>EMO X</td>
<td>Jonathan Knudsen</td>
<td>Defensics</td>
<td>Synopsys advisory</td>
</tr>
<tr>
<td>CVE-2021-33176</td>
<td>BSDA-2021-1609</td>
<td>VerdeMQ</td>
<td>Jonathan Knudsen</td>
<td>Defensics</td>
<td>Synopsys advisory</td>
</tr>
<tr>
<td>CVE-2021-33430</td>
<td>BSDA-2021-1716</td>
<td>Zephyr Project</td>
<td>Matias Karhuma</td>
<td>Defensics</td>
<td>Synopsys advisory</td>
</tr>
<tr>
<td>CVE-2021-33431</td>
<td>BSDA-2021-1718</td>
<td>Zephyr Project</td>
<td>Matias Karhuma</td>
<td>Defensics</td>
<td>Synopsys advisory</td>
</tr>
<tr>
<td>CVE-2021-33432</td>
<td>BSDA-2021-1727</td>
<td>Zephyr Project</td>
<td>Matias Karhuma</td>
<td>Defensics</td>
<td>Synopsys advisory</td>
</tr>
<tr>
<td>CVE-2021-33433</td>
<td>BSDA-2021-1734</td>
<td>Zephyr Project</td>
<td>Matias Karhuma</td>
<td>Defensics</td>
<td>Synopsys advisory</td>
</tr>
</tbody>
</table>
What about SAST?

**UNSAFE_DESERIALIZATION** finds unsafe deserialization injection vulnerabilities, which arise when uncontrolled dynamic data is used within an API that can deserialize or unmarshall an object. This security vulnerability might allow an attacker to bypass security checks or to execute arbitrary code.

SAST can detect unsafe and untrusted deserialization, but the open-source component source code must be included in the analysis, not just the classes.

Coverity included jackson-databind-2.4.3.jar (only class files) which were used for the analysis of the source captured, but not part of the captured code which contained this issue.

The **DISTRUSTED_DATA_DESERIALIZATION** checker reports an issue any time distrusted data is passed into a deserialization API. An attacker who can control the deserialized object might be able to subvert aspects of the application functionality. This audit mode checker flags these code patterns for review.
Software Composition Analysis

Summary

1. Use the type of scanning necessary to produce the results needed (detector, signature, binary, snippet, etc.) - some require manual validation

2. Use policies to prioritize, someone has already done the ethical hacking and risk assessment for you!

3. When a quick component update is not practical, use the research provided to understand the risk and mitigation
Today’s Agenda

1. Software Development Process
2. SAST
3. SCA
4. DAST (IAST/Fuzz)
5. ASOC
6. Help!
Ethical Hacking (DAST)

Many tools may be used to automate ethical hacking, sending compromising data to an application and reviewing responses.

Products like IAST deploy agents to watch and validate issues as web applications are exercised.
The password used via the JDBC API to access the local database was weak.

A weak database password allows lateral hacking from a compromised host to accessible databases.
What about SAST?

Dynamic testing *may* not have been required to detect this issue.

But SAST only detected the hardcoded password in a test program (also a no-no).

The password is hardcoded in `appContext-jdbc.xml` which is ok if the file is encrypted and access is limited.
Fuzzing

Applying additional automation, a fuzzer may send all kinds of data to the application endpoints.

In combination with DAST, this is a powerful detection system.
15 new issues detected!

Clickjacking, also known as "User Interface Redressing" is a security vulnerability that exposes the business to risks of having application users unknowingly perform various operations using the browser. Operations such as submittingクレジットカード情報 would seem like they're using the frame.

In order to block attempts for this attack, the web site should deny unpermited entities to host the web site inside a frame.

It is recommended that the web site will embed the headers 'X-Frame-Options' and 'Content-Security-Policy' in every HTTP response sent back from the server. These headers instruct browsers to enforce improper usage of HTML frame elements that might lead to this attack.

For instance, the following headers will instruct browsers to deny hosting of web site pages inside HTML frame elements:

```
X-Frame-Options: DENY
Content-Security-Policy: frame-ancestors 'none'
```

If it is needed to host pages of the web sites inside HTML frame elements, it is recommended to use the following header values to allow such hosting only for the same origin:

```
X-Frame-Options: SAMEORIGIN
Content-Security-Policy: frame-ancestors 'self'
```

If for some reason it is not possible to add the above headers, an alternative solution could be a javascript blockage of frame hosting. Following is a sample javascript code portion that blocks hosting of web pages inside frames:

```
if (top.location !== self.location) {
    top.location = self.location;
}
```

Note: In some legacy browsers such as IE 7 or Safari 4.0.4, the above solution might be needed.
Dynamic Application Security Testing

Summary

1. Automate whenever possible
2. Results may take investigation
3. Correlation with other tools is beneficial
Today’s Agenda

1. Software Development Process
2. SAST
3. SCA
4. DAST (IAST/Fuzz)
5. ASOC
6. Help!
<table>
<thead>
<tr>
<th>ID</th>
<th>Type</th>
<th>Tool</th>
<th>CWE</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>17225</td>
<td>Using Components with Known Vulnerabilities</td>
<td>68 active results from Black Duck Hub</td>
<td>1352</td>
<td>Components/jackson-databind:2.4.3</td>
</tr>
<tr>
<td>17229</td>
<td>Credentials Management</td>
<td>Seeker / Weak Password Used in Database Connection</td>
<td>255</td>
<td>-</td>
</tr>
<tr>
<td>17224</td>
<td>Data Structure</td>
<td>Coverity / SIGMA.cbc_insecure_padding</td>
<td>461</td>
<td>UserController.java:141</td>
</tr>
<tr>
<td>17226</td>
<td>Credentials Management</td>
<td>Coverity / Medium impact security / Hardcoded Credentials</td>
<td>255</td>
<td>UserController.java:143</td>
</tr>
<tr>
<td>17220</td>
<td>Credentials Management</td>
<td>Coverity / Medium impact security / Hardcoded Credentials</td>
<td>255</td>
<td>BCryptPasswordEncoderTester.java:13</td>
</tr>
<tr>
<td>17228</td>
<td>Clickjacking</td>
<td>Seeker / Clickjacking</td>
<td>1021</td>
<td>/spring-angularjs-java-webapp-template-project-1.0-SNAPSHOT</td>
</tr>
<tr>
<td>17227</td>
<td>Clickjacking</td>
<td>Seeker / Clickjacking</td>
<td>1021</td>
<td></td>
</tr>
<tr>
<td>17240</td>
<td>Logging</td>
<td>Coverity / Audit impact security / Log injection</td>
<td>1210</td>
<td>Category</td>
</tr>
<tr>
<td>17231</td>
<td>Logging</td>
<td>Coverity / Audit impact security / Log injection</td>
<td>1210</td>
<td>Category</td>
</tr>
<tr>
<td>17230</td>
<td>Logging</td>
<td>Coverity / Audit impact security / Log injection</td>
<td>1210</td>
<td>Category</td>
</tr>
<tr>
<td>17228</td>
<td>Logging</td>
<td>Coverity / Audit impact security / Log injection</td>
<td>1210</td>
<td>Category</td>
</tr>
<tr>
<td>17227</td>
<td>Logging</td>
<td>Coverity / Audit impact security / Log injection</td>
<td>1210</td>
<td>Category</td>
</tr>
<tr>
<td>17216</td>
<td>Logging</td>
<td>Coverity / Audit impact security / Log injection</td>
<td>1210</td>
<td>Category</td>
</tr>
<tr>
<td>17215</td>
<td>Logging</td>
<td>Coverity / Audit impact security / Log injection</td>
<td>1210</td>
<td>Category</td>
</tr>
<tr>
<td>17212</td>
<td>Logging</td>
<td>Coverity / Audit impact security / Log injection</td>
<td>1210</td>
<td>Category</td>
</tr>
<tr>
<td>17211</td>
<td>Logging</td>
<td>Coverity / Audit impact security / Log injection</td>
<td>1210</td>
<td>Category</td>
</tr>
<tr>
<td>17221</td>
<td>Logging</td>
<td>Coverity / Audit impact security / Log injection</td>
<td>1210</td>
<td>Category</td>
</tr>
<tr>
<td>17206</td>
<td>Logging</td>
<td>Coverity / Audit impact security / Log injection</td>
<td>1210</td>
<td>Category</td>
</tr>
<tr>
<td>17204</td>
<td>Logging</td>
<td>Coverity / Audit impact security / Log injection</td>
<td>1210</td>
<td>Category</td>
</tr>
<tr>
<td>17203</td>
<td>Logging</td>
<td>Coverity / Audit impact security / Log injection</td>
<td>1210</td>
<td>Category</td>
</tr>
<tr>
<td>17186</td>
<td>Logging</td>
<td>Coverity / Audit impact security / Log injection</td>
<td>1210</td>
<td>Category</td>
</tr>
</tbody>
</table>

**Issue Density**

**Design Concern**

**Lateral Hacking**

**Technical Debt**

**Clickjacking**

**Zero-Click RCE**

**Oracle Attack**
Today’s Agenda

1. Software Development Process
2. SAST
3. SCA
4. DAST (IAST/Fuzz)
5. ASOC
6. Help!
Thank You!

Rich Newman
rnewman@synopsys.com
949.466.5283